웹시스템설계 실습문서 Lab 09 (과제 2)

최지헌 unidev@ajou.ac.kr
이재현 dlwogus8888@ajou.ac.kr

2024-11-19

목차

1 MongoDB

- MongoDB란 Document 기반의 No-SQL Database이다.
- Document는 MongoDB의 데이터 저장 단위로, 한 개 이상의 Key-Value 쌍으로 이루어진 객체를 BSON 형태로 저장한다.
- SQL expression이 존재하지 않으며, 모든 데이터 조작은 mongoshell 또는 ODM(Object Data Mapping) 라이브러리에서 메소드 호출 방식으로 동작한다. Schema-free한 데이터 저장 방식 덕분에 같은 Collection에 속한 Document라도 다른 유형의 데이터 필드를 가질 수 있다 모든 Document는 기본적으로 **_id** 라는 고유한 값을 가진 필드가 존재한다.

Document는 MongoDB의 데이터 저장 단위로, 한 개 이상의 Key-Value 쌍으로 이루어진 객체를 BSON 형태로 저장한다. **Collection**은 데이터를 용도에 따라 분류하기 위한 Document의 모음이며, 모든 Document는 특정 Collection에 속한다. (RDBMS의 Table과 유사함) **Database**는 여러 Collection들을 담고 있는 물리적 저장소이다. MongoDB 서버는 여러 개의 Database를 가질 수 있으며, "System" 은 서버 운영에 필요한 메타 정보를 저장하기 위해 예약되어 있으므로 사용 불가하다.

1.1 Mongoose

Mongoose는 Node.js 기반의 MongoDB ODM 라이브러리이다. MongoDB Driver가 기본적으로 제공하는 CRUD 기능을 포함하여 다양한 추가 기능 제공한다.

- Schema (https://mongoosejs.com/docs/guide.html)
  - Schema 기반의 Collection 데이터 모델링이 가능하고, 데이터를 DB에 저장하기 전에 검사 가능
- Population (https://mongoosejs.com/docs/populate.html)
  - Document B를 참조(reference)하고 있는 Document A를 쿼리할 때, 참조 위치에 실제 Document B의 데이터가 치환되어 값이 반환되는 기능
- Callback-base / Promise-base 방식을 모두 지원하는 유연한 API 제공

1.2 Mongoose를 사용해 MongoDB 연결

Mongoose는 NPM을 통해 프로젝트에 추가할 수 있다.

```
$ npm install mongoose
```

MongoDB 서버와 통신하기 위한 connection 객체를 생성하여 연결한다.

```
async function connect(){
await mongoose.connect('mongodb://127.0.0.1:27017/test_db',
                    { useUnifiedTopology: true, useNewUrlParser: true })
}

connect();
```

서버 주소는 mongodb://ADDRESS:PORT/DATABASE의 형태로 기술한다. mongoDB는 따로 설정을 수정하지 않았다면 27017번 포트로 접근할 수 있다. 같은 컴퓨터 내에 설치되어 있는 mongoDB에 접근한다면 주소로 127.0.0.1이나 localhost를 사용한다.

1.3 Schema 정의

스키마(schema)란 collection에 저장할 document의 데이터 구조를 모델링하는 기능이다.

```
const userSchema = new mongoose.Schema({
    name: {
        first: String,
        last: String
    },
    createdAt: {
        type: Date,
        default: Date.now
    }
})
```

필드 이름과 속성으로 구성되며 속성으로 타입만 정의한다면 하위 객체를 넣을 필요 없이 바로 타입값을 넣을 수 있다. 사용 가능한 속성들로는 다음과 같은 것들이 있다.

- type: 이 필드의 타입을 지정한다. 값으로 자바스크립트의 타입 객체들이 온다.
- unique: 해당 collection 내에서 document 간 중복되지 않는 필드임을 명시한다. 이 필드가 중복되는 상황을 만드는 document 추가나 수정은 에러가 발생하게 된다. 값으로 boolean값이 온다.
- required: 필수값임을 명시한다. 이 값 없이 document를 생성한다면 에러가 발생한다. 값으로 boolean 값이 온다.
- default: document를 생성하여 저장할 때 이 필드의 값이 존재하지 않는다면 기본적으로 주어질 기본값을 정의한다. 값으로 아무것이나 올 수 있다.

1.4 Model 생성

모델(model)이란 앞서 정의한 스키마를 기반으로 collection에 조작을 가하기 위한 인터페이스 객체이다.

```
// "User" Collection에 데이터 조작(CRUD)을 처리하기 위한 모델 생성
const userModel = mongoose.model('User', userSchema)
```

여기서 model 메서드의 첫 번째 파라미터로 주어진 "User"가 collection의 이름이 된다. 단수로 지정할 경우 자동으로 s가 붙어 복수 형태로 collection 이름이 지정된다.

1.5 CRUD Example (Create)

```
const data = await userModel.create({ name: { first: "aa", last: "bb" } })
const res = await data.save();
console.log(res); //새로 생성된 document를 볼 수 있다.
```

1.6 CRUD Example (Read)

```
const result = await userModel.find({name: { first: "aa", last:"bb" }}); // first가 aa
    이고 last가 bb인 모든 document 반환
```

1.7 CRUD Example (Update)

조건에 맞는 모든 document의 필드값을 갱신할 수 있다. updateMany(조건, 갱신값)의 형태로 기술된다.

```
const result = await userModel.updateMany({name: { first: "aa", last:"bb" }}, {name: {last:
    "cc"}}); //first가 aa이고 last가 bb인 모든 doucment에 대해 last를 cc로
    갱신
```

응답 객체로 필드값으로 n과 nModified, ok가 담긴 객체가 반환된다. n은 선택된 document의 총 개수, nModified는 실제로 수정된 document의 개수이며 ok는 성공 여부이다. 조건에 일치하는 것 중 첫 번째 것만 업데이트하기 위한 메서드로 updateOne이 있다. 갱신값에는 기술된 필드만 갱신되고 기술되지 않은 필드는 기존값 그대로 남는다. 완전히 document를 덮어씌우려면 replaceOne를 사용한다.

1.8 CRUD Example (Delete)

조건에 맞는 모든 document를 제거한다. deleteMany(조건)의 형태로 기술된다.

```
const result = await userModel.deleteMany({name: { first: "aa", last:"bb" }}) // first가 aa
    이고 last가 bb인 모든 document 삭제
```

조건에 일치하는 것 중 첫 번째 것만 제거하는 api로 deleteOne이 존재한다.

2 실습 과제

2.1 프로젝트 디렉토리 구조

```
Lab09_(본인학번)/
├── frontend
│   ├── (생략)
│   ├── src
│   │   ├── (생략)
│   │   ├── components
│   │   │   ├── TodoFilter.js
│   │   │   ├── TodoInput.js
│   │   │   ├── TodoItem.js
│   │   │   └── TodoList.js
│   │   ├── App.css
│   │   ├── App.js
│   │   └── api.js
├── backend
│   ├── (생략)
│   └── main.js
```

※ frontend 디렉토리는 "npx create-react-app frontend" 스크립트로 생성된 디렉토리이다.
※ 기존 ./frontend/src/App.css, ./frontend/src/App.js 를 bb에 제공된 App.css, App.js로 각각 대체한다 (replace)
※ ./frontend/src에 components 폴더를 생성 후 bb에 제공된 TodoFilter.js, TodoInput.js, TodoItem.js, TodoList.js를 해당 components 폴더에 추가한다.
※ ./frontend/src/App.css와 ./frontend/src/App.js 및 ./frontend/src/components 내부 파일들은 제공된 그대로 사용하며, 수정하지 않는다.
※ node_modules 디렉토리를 삭제한 뒤 압축하여 제출한다.

2.2 프로그램 구현 내용

Lab 08에서 react 만으로 구현한 Todo Application에 대하여 MongoDB를 활용한 데이터베이스를 Express 서버로 관리하며, React 클라이언트에서 이를 연동하여 필터링, 추가, 우선순위 변경, 삭제, 완료 상태 토글 등의 기능을 구현한다.

2.3 사전 정보

다음 파일들은 사전에 제공된 것으로, 수정하지 않는다.

- ./frontend/src/components
  - Todo Application의 UI 요소들을 포함한다.
  - Lab 08에서의 요구 사항에 맞게 작성된 TodoInput, TodoItem, TodoList, TodoFilter로 구성돼 있다.

  - 각 컴포넌트는 Todo 데이터를 표시, 필터링, 추가, 삭제, 완료 상태 토글 등의 기능을 담당한다.

- ./frontend/src/App.css
  - 웹 페이지 스타일링을 정의한 css 파일이다.
  - 스타일링이 모두 적용된 화면은 Figure 1과 같다.

- ./frontend/src/App.js
  - 다음과 같은 역할을 담당한다.
    * 다음 4가지를 상태 관리하여 화면 상에 반영
      · todos(현재 화면에 표시할 Todo 항목들의 리스트)
      · statusFilter(상태 필터의 값으로, 완료, 진행 중, 전체 중 하나를 선택)
      · priorityFilter(우선순위 필터의 값으로, high, medium, low 중 하나 이상 선택 가능)
      · selectedTodos(사용자가 체크박스를 통해 선택한 Todo의 ID 배열로, 선택된 항목들을 완료/미완료 상태로 변경하거나 삭제 작업을 수행할 때 사용)
    * ./frontend/src/api.js를 통해 서버와 통신
    * ./frontend/src/components 내 정의된 컴포넌트들을 통합하여 웹 화면을 구성
    * 하단에 완료/진행 중 토글 및 등록된 Todo 삭제 버튼 구성

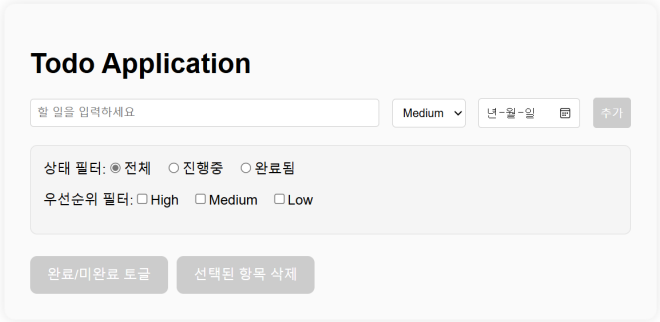

Figure 1: 초기 화면

2.3.1 구현 요구사항

./frontend/src/api.js

- api.js는 서버로 HTTP 요청 전송 및 응답 처리 로직을 담당한다.
  - getTodos
    * server의 /api/todos 엔드포인트에 GET 요청을 보내 Todo 목록을 받아온다.
    * 필터링 조건과 대응되는 상태와 우선 순위를 쿼리 매개 변수로 전달한다.
    * 필터링 조건에서 각각을 선택 및 미선택 모두 가능하므로 상태와 우선 순위는 optional 하다.

    * 아래는 server로 요청되는 예시 API 엔드포인트 및 쿼리 스트링이다.

```
/api/todos?status=progress&priorities=low
```

    * 다음을 참고하여 사용 맥락을 파악한다.
        · App.js의 fetchTodos
        · App.js 내 useEffect(30 - 32 lines)

- addTodo
    * server의 /api/todos 엔드포인트에 POST 요청을 보내 Todo가 새로 등록될 수 있도록 한다.
    * body에 제목, 우선 순위, 마감일을 아래와 같은 JSON 형식으로 포함하여 전달한다.

```
{
   "text": "study",
   "priority": "high",
   "dueDate": "2024-11-20"
}
```

    * 다음을 참고하여 사용 맥락을 파악한다.
        · App.js의 handleAddTodo
        · components 폴더 내 TodoInput.js

- updateTodo
    * server의 /api/todos/:id 엔드포인트에 PUT 요청을 보내 특정 id에 대응되는 Todo 정보를 업데이트한다.
    * body에는 Todo의 수정할 정보를 포함한다. 아래는 우선 순위를 medium으로 수정하는 상황에서의 예시 형식이다.

```
{
   "priority": "medium",
}
```

    * 다음을 참고하여 사용 맥락을 파악한다.
        · App.js의 updateTodoPriority
        · components 폴더 내 TodoList.js

- deleteTodos
    * server의 /api/todos 엔드포인트에 DELETE 요청을 보내 특정 Todo들을 삭제한다.
    * body에는 삭제할 항목들의 id 배열을 포함한다. 아래와 같이 key를 "ids", value를 id 배열로 한다.

```
{
    "ids": ["673acc259d0d793ca843cd06", "673ad74434e1fb54bd983513"]
}
```

    * 다음을 참고하여 사용 맥락을 파악한다.
        · App.js의 handleDeleteTodos
        · App.js의 삭제 버튼(137 - 143 lines)
  - toggleTodosStatus
    * server의 /api/todos/toggle-status 엔드포인트에 PATCH 요청을 보내 여러 Todo의 완료 상태를 토글한다.
    * body에는 수정할 Todo들의 id 배열을 아래와 같이 포함한다.

```
{
    "ids": ["673afca834e1fb54bd983524", "673ad74434e1fb54bd983513"]
}
```

    * 다음을 참고하여 사용 맥락을 파악한다.
        · App.js의 updateTodoPriority
        · components 폴더 내 TodoList.js, TodoItem.js

./backend/main.js

- main.js는 Express 서버로, MongoDB와 연결되어 client의 요청을 처리 및 응답한다.
  - GET /api/todos
    * client로부터 전달받은 필터링 조건에 따라 Todo 목록을 반환한다.
    * 필터링 조건이 없는 경우 전체 Todo를 반환한다.
    * 아래와 같은 형식으로 반환한다.

```
{
    "_id": "673ad74434e1fb54bd983513",
    "text": "Example Todo",
    "completed": true,
    "priority": "high",
    "dueDate": "2024-12-31",
    "createdAt": "2024-11-01"
}
```

  - POST /api/todos
    * 새 Todo를 DB에 추가한다.
    * client로부터 전달된 요청의 body 형식은 addTodo를 참고한다.
    * 저장된 Todo를 클라이언트에 반환한다.
  - PUT /api/todos
    * 특정 Todo 항목의 정보를 수정한다.
    * client로부터 전달된 요청의 body 형식은 updateTodo를 참고한다.

    - 업데이트된 Todo를 클라이언트에 반환한다.
- DELETE /api/todos
    - 요청된 Todo 항목들의 ID 배열에 해당하는 Todo 항목들을 삭제한다.
    - client로부터 전달된 요청의 body 형식은 deleteTodos를 참고한다.
- PATCH /api/todos/toggle-status
    - 요청된 Todo 항목들의 완료 상태를 각각 토글한다. 즉, 완료인 것은 진행 중인 것으로, 진행 중인 것은 완료 상태로 바꾼다.
    - client로부터 전달된 요청의 body 형식은 toggleTodosStatus를 참고한다.
    - 업데이트된 Todo 목록을 클라이언트에 반환한다.
- Todo의 스키마는 아래와 같이 정의한다.

```
{
  "text": "string (Todo 제목 / 필수값)",
  "completed": "boolean (완료 여부 / 기본값 : false)",
  "priority": "string (우선 순위 / 기본값 : medium /
     가능값 : high, medium, low)
  "dueDate": "string (마감일 / 형식 예시 : "2024-11-25")",
  "createdAt": "string (생성일 / 형식 예시 " "2024-11-25")"
}
```

- Express 서버 port는 8000 이어야한다.
- MongoDB 연결 주소는 mongodb://localhost:27017/todolist 이어야한다.

2.4 Submission

제출 기한: 12월 01일 23:59 / 이번 과제의 지각 제출은 받지 않는다. (최소한 과제 기한 1시간 전에 제출하는 것을 권장함)